FYS-STK4155 - PROJECT 2: CLASSIFICATION AND REGRESSION ANALYSIS

ALEXANDER HAROLD SEXTON
ERIK JOHANNES HUSOM

NOVEMBER 13, 2019

Abstract. In this project we study the predictive capabilities of logistic regresson and artificial neural networks on a binary classification problem. The methods are applied on a data set from a bank in Taiwan, where we aim to predict credit card holders default payments. Our findings have been compared with those of a recent scientific article, which uses different machine learning techniques applied to the same data set. We found neural networks to give the highest accuracy, with a score of 0.86, where 0.64 of the cases were classified as true positives and 0.92 true negatives. We also use neural networks for linear regression on a set of generated terrain data, and compare its performance to the methods of ordinary least squares, ridge and lasso. Here we find that our neural network regressor, which gave a mean squared error of 0.0415 on the test set, does not perform equally well as the more traditional regression methods, for example the Ordinary Least Squares method, which gave a test mean squared error of 0.0136.

Contents

1. Introduction

In machine learning, classification is a way of dealing with problems in which we wish to indentify which predefined class a set of observations can be categorized as. Typical applications of such methods are for example diagnostics in the field of medicine, where based upon the symptoms of the patient, one can use a trained model to classify what kind of illness or disease the patient has, or it can be applied to training computers to recognize patterns or features in images, such as hand written digits or letters, so that they can automatically be classified. A commonly used method for classification is Logistic regression, which aims to model the probability that a set of observations belongs to a specific class or leads to an outcome. Another method in machine learning which is often used in classification is Artificial neural networks, which is a powerful tool that can adapt to many different problems. This method is inspired by biological neurons, and can be scaled to model both simple and very complex phenomenons. Even though a neural network can be very versatile and give precise predictions, it is harder to obtain any insights about the relation between the features and the targets of the data set, compared to more traditional methods like logistic regression. Artificial neural networks are very adatable, so they are not only used for classification, but can also work with regression problems.

In the first part of this project, we apply both of these methods on a data set from a bank in Taiwan, containing the information of credit card holders and whether they default on their payments or not, and aim to train our models in predicting future defaults. In our study, we have compaired our results with those of a recent research article[1] that has looked at different classification methods applied to the same data set. In the second part, we use neural networks on a regression problem, and compare its performance with linear regression methods, which we have studied in an earlier project[2]. An other aspect of this project is to explore the different ways of preprocessing the data which we are given, as this can have a big impact on how well the models perform.

In this report, we include the background theory of the methods we have used in the project, as well as a short review of how we have implemented the methods. Our findings are presented in the results section, followed by a discussion of the results and lastly a conclusion.

2. Methods

2.1. **Logistic Regression.** Logistic regression is a method of modeling the probability that a set of input variables $\boldsymbol{x}$ leads to an outcome or class y_i, $i = 1, 2, \dots K$, where K is the number of possible classes. The model uses the logistic (or Sigmoid) function, which in the binary case with only two classes

$y_i \in [0,1]$, is given by

$$p(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}} \tag{1}$$

With instead a set of input variables $\boldsymbol{x}$, we can express the probabilities of each of the two outcomes as

$$p(y_i = 1|x_i, \boldsymbol{\beta}) = \frac{\exp\left(\boldsymbol{\beta}^T x_i\right)}{1+\exp\left(\boldsymbol{\beta}^T x_i\right)} \tag{2}$$

$$p(y_i = 0|x_i, \boldsymbol{\beta}) = 1 - p(y_i = 1|x_i, \boldsymbol{\beta}), \tag{3}$$

where $\boldsymbol{\beta}$ are the coefficients which we want to estimate with our data. Defining the set of all possible outcomes in our data set $\mathcal{D} = (x_i, y_i)$, and assuming that these samples are independent and identically distributed, the total likelihood for all possible outcomes in $\mathcal{D}$ can be approximated by the product of the individual probabilities[3][p.120] of a specific outcome y_i:

$$P(\mathcal{D}|\boldsymbol{\beta}) = \prod_{i=1}^{n} \left[p(y_i = 1|x_i, \boldsymbol{\beta}\right]^{y_i} \left[1 - p(y_i = 1|x_i, \boldsymbol{\beta})\right]^{1-y_i} \tag{4}$$

Since in our model we aim to maximize the probability of seeing the observed data, we use the Maximum Likelihood Estimation principle, which by taking the log of the above equation leads to the log-likelihood function in $\boldsymbol{\beta}$:

$$\log P(\mathcal{D}|\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \log p(y_i = 1|x_i, \boldsymbol{\beta}) + (1-y_i)\log(1 - p(y_i = 1|x_i, \boldsymbol{\beta}))\right] \tag{5}$$

By taking the negative of the above expression and re ordering the logarithms, we arrive at what is known as the cross entropy:

$$\mathcal{C}(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \left[y_i \boldsymbol{\beta}^T x_i - \log\left(1 + \exp\left(\boldsymbol{\beta}^T x_i\right)\right)\right], \tag{6}$$

which we use as our cost function for logistic regression. Minimizing the cross entropy is the same as maximizing the log-likelihood:

$$\frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \beta} = -\sum_{i=1}^{n} x_i (y_i - p(y_i = 1|x_i, \boldsymbol{\beta})) = 0, \tag{7}$$

and computing the second derivative of this quantity gives

$$\frac{\partial^2 \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = \sum_{i=1}^{n} x_i x_i^T p(y_i = 1|x_i, \boldsymbol{\beta})(1 - p(y_i = 1|x_i, \boldsymbol{\beta})). \tag{8}$$

Defining the diagonal matrix $\boldsymbol{W}$ with elements $p(y_i = 1|x_i, \boldsymbol{\beta})(1 - p(y_i = 1|x_i, \boldsymbol{\beta})$, $\boldsymbol{X}$ as the design matrix containing the data, $\boldsymbol{y}$ as the vector with our y_i values and finally $\boldsymbol{p}$ as the vector of fitted probabilities, we can express the first and second derivatives in matrix form:

$$\frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \beta} = -\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{p}) \tag{9}$$

$$\frac{\partial^2 \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X}, \tag{10}$$

also known as the Jacobian and Hessian matrices. The section below 2.2 explains how we in practice solve this equation for $\boldsymbol{\beta}$ to obtain the optimal parameters of the predictors in the data set.

2.2. **Gradient Descent.** A popular iterative method to finding the roots of an equation is the *Newton-Ranphson method*[4][p. 116], which is derived from Taylor expanding a function f in x sufficiently close to the solution x_0. In the one-dimensional case it is expressed as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{11}$$

In logistic regression, the function f is substituted with the minimized expression for the cross entropy (eq. 7). We then have that the values $\boldsymbol{\beta}$ which maximize the log-likelihood can be found iteratively by

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - \left(\frac{\partial^2 \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T}\right)^{-1} \left(\frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right) \tag{12}$$

$$= \boldsymbol{\beta}^n - \left(\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X}\right)^{-1} \left(-\boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{p})\right), \tag{13}$$

where we have inserted the expressions for the Jacobian and Hessian matrices of the cross entropy (eq. 9).

In optimization problems where computing the inverse of a matrix is impossible or simply too computationally demaning, using the Newton-Raphson method as is, is not a viable option. Looking back at the expression in equation 12, one approach is to simply replace the inverse Hessian with a parameter η, referred to as the *learning rate*. This leads to the method of *steepest descent*[4][p. 52], which is based on the observation that if a function $F(\boldsymbol{x})$ is differentiable in the neighbourhood of a point $\boldsymbol{a}$, the function decreases the fastest in the direction of the negative gratient $-\nabla F(\boldsymbol{a})$. It then follows that the iterative method for finding the optimal $\boldsymbol{\beta}$ values in the logistic regression case, can be expressed as

$$\boldsymbol{\beta}_{n+1} = \boldsymbol{\beta}^n - \eta \nabla \mathcal{C}(\boldsymbol{\beta}) = \boldsymbol{\beta}^n - \eta \left(\boldsymbol{X}^T (\boldsymbol{p} - \boldsymbol{y})\right) \tag{14}$$

For a convex function and a small enough value for η, it is obvious that this method always converges to the global minimum, but its limitation is that it is sensitive to the initial conditions of $\boldsymbol{\beta}$ and may therefore get stuck in a local minima when dealing with a multivaried non-convex function. To avoid the shortcomings of the gradient descent method, we introduce an element of stochasticity in the way we calculate the gradients $\nabla \mathcal{C}(\boldsymbol{\beta})$. Instead of calculating the gradients on our whole data set $\boldsymbol{X}$, we randomly spilt the data into M minibatches, and calculate the gradient for each minibatch. Denoting each minibatch as B_k where $k = 1, 2, ..., n/M$, the new approximation to the gradient is

$$\nabla \mathcal{C}(\boldsymbol{\beta}) = \sum_{i \in B_k}^{n/M} \nabla c_i(\boldsymbol{x_i}, \boldsymbol{\beta}), \tag{15}$$

where k is chosen randomly with equal probability $[1, n/M]$.

2.3. **Neural Networks.** *Artificial neural networks* (ANN) is a machine learning technique inspired by the networks of neurons that make up the biological brain. There are several different types of neural networks, and in this project we use a multi-layer perceptron (MLP), where there is one or more *hidden layers* between the input and output layers, and each hidden layer consists of several *neurons/nodes*. The signal will only go in one direction, from the input layer to the output layer, which makes this a *feedforward neural network* (FNN). Additionally, all nodes of a layer will be connected to every node on the previous layer, which makes it a *fully connected layer*. In this section we will review the essential parts of how a neural network functions.

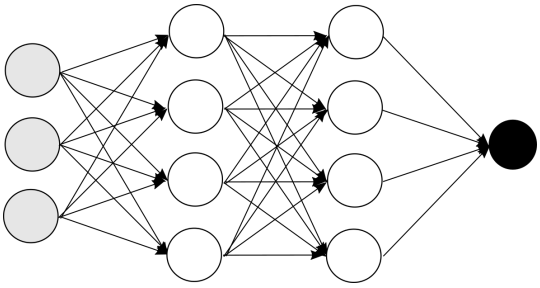

FIGURE 2.1. Schematic neural network. The circles represent nodes, and the colors indicate what layer they are a part of: Grey means input layer, white means hidden layer and black means output layer. The arrows show that all nodes of one layer is connected to each of the nodes in the next layer; i. e. we have only fully connected layers.

2.3.1. *Feed-forward.* The basic idea of this kind of network is that the input information (i. e. the features/predictors of our data set) y, are passed through all of the nodes in the hidden layers, until it ends up in the output layer. Figure 2.1 shows an example with three inputs, two hidden layers with four nodes each, and one output. This process is called *forward feeding* of the network. Each input is multiplied by a weight w_i when passed into a node, and the result is called z_i. The signal is then evaluated by an activation function $a(z_i) = a_i$, which then becomes the output from each node. Since we are dealing with several observations, we assemble the weights w_i into a matrix W. Usually we add a bias b_i to each of the nodes in a hidden layer, to prevent outputs of only zeros. The output a_l of a layer l then becomes (where a_{l-1} is the output from the previous layer):

$$a_l = a(a_{l-1}W_l + b_l) \tag{16}$$

In the case of the first hidden layer, the input matrix X, which contains the features of our data set, will take the place of a_{l-1}. The weights are usually initialized with random values, for example using a normal or uniform distribution, because if all the weights were the same, all nodes would give the same output. In this project we use a standard normal distribution when doing classification, but in the case for regression we use an initialization proposed by Xavier Glorot and Yoshua Bengio[5], where we scale the randomly distributed weights by a factor $1/\sqrt{n_{l-1} + n_l}$ (where n_{l-1} is the size of the previous layer, and n_l is the size of the current layer). The biases are given a small non-zero value, in our case $b_i = 0.01$ for all layers, in order to ensure activation.

2.3.2. *Activation functions.* The choice of activation function may have a huge effect on the model, and there are several options that may work depending on what data set we want to process, how it is scaled and if it is a regression or classification case. In a feed-forward neural network, we must have activation functions that are non-constant, bounded, monotonically-increasing and continous for the network to function correctly on complex data sets. It is possible to choose different activation functions for each hidden layer, but in this project we have the same activation function throughout the networks we use.

Common choices for activation functions are the logistic/sigmoid function, the Rectified Linear Unit function (ReLU) and the tanh-function[6][p. 288]. For classification problems, the sigmoid function, presented in equation 1, is often preferred. This function gives only output that is between 0 and 1, which is good when we want to predict the probability as output of the network. In multiclass classification

problems the so-called softmax function is a better choice, because it forces the sum of probabilites for the possible classes to be 1, but for binary classification problems, like we have in this project, the sigmoid function is sufficient.

For regression we use another common activation function, the ReLU function:

$$\text{ReLU} = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{17}$$

2.3.3. *Output layer and back propagation.* After the feed-forward process is done, the output layer will contain the predictions of the neural network, which we will compare with the true targets of our training data set. Based in this comparison, we will adjust the weights and biases of the network in order to improve the performance. Since the weights and biases usually are initialized randomly, the first feed-forward iteration will most likely give very wrong results. One feedforward pass is usually called an *epoch*, and we choose the number of epochs based on how much "training" the network needs in order to give satisfactory results. The process of adjusting the weights and biases is called *back propagation*.

The comparison between the output from the network $\hat{y}$ and the true targets of our training data set y is done with a predefined cost function $\mathcal{C}$. We want to minimize the cost, and to do this we need to know how the weights and biases must be adjusted to obtain a better result. This is done by calculating the gradient of the cost function, and the exact derivation depends on how the cost function is defined. Based on the derivative of the chosen cost function, and by using the chain rule[3][p. 396], one can show that in general terms the expression for the output error δ_L becomes

$$\delta_L = \frac{\partial \mathcal{C}}{\partial a_L} \frac{\partial a_L}{\partial z_L}, \tag{18}$$

and the back propagate error for the other layers $l = L-1, L-2, ..., 2$ is

$$\delta_l = \delta_{l+1} W_{l+1}^T a'(z_l), \tag{19}$$

where a' is the derivative of the activation function. The update of the weights and biases for a general layer l is calculated by

$$W_l \leftarrow = W_l - \eta a_{l-1}^T \delta_l, \tag{20}$$

$$b_l \leftarrow = b_l - \eta \delta_l, \tag{21}$$

where η is the learning rate, which specifies how much the weights and biases should be adjusted for each back propagation. After the update of the weights and biases, we start a new epoch with another forward-feed of the network. We utilize minibatches, as described in the section above for gradient descent, in order to reduce the computational cost of the algorithm. The learning rate which is passed into the method is divided by the chosen batch size, to account for differences in the data size that is passed through each iteration.

In our classification problem, we use the binary cross-entropy as a cost function:

$$\mathcal{C}_{\text{classification}} = -(y \log(p) + (1-y)\log(1-p)), \tag{22}$$

where p is the predicted probability of an observation, and y is the true target, which will be either 0 or 1 since we have a binary classification problem. The probabilities p is the same as the output for our network $\hat{y}$. Combining this with the sigmoid as the activation function in the last layer, we use equation 18 to get the formula for the output error:

$$\delta_L = \hat{y} - y. \tag{23}$$

For regression we use a slightly modified version of the mean squared error:

$$\mathcal{C}_{\text{regression}} = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2. \tag{24}$$

The factor $\frac{1}{2}$ is used to simplify the derivative of the cost function, because if we use the identity function (which returns the same values that is the input to the function, i. e. we have no activation function) as an activation function in the last layer, the output error is the same as in equation 23. This is very convenient, because we can use the same calculation at the beginning of each back propagation whether we are doing classification or regression. A common practice when using neural networks on regression problems is to use the ReLU as activation function in the hidden layers, and then use the identity function in the last layer[6][p. 290].

2.4. **Error Metrics.** For classification, we use the accuracy score to measure how well our models perform, which is formulated as

$$\text{accuracy} = \frac{1}{n}\sum_{i=0}^{n} I(t_i = y_i). \tag{25}$$

where t_i represents the target output, y_i represents the output from our model, and I is the indicator function, which returns 1 if $y_i = t_i$ and 0 otherwise.

We also make use of the so-called *confusion matrix*, which is another commonly used error metric in classification problems. Instead of measuring only in how many instances the model has made a correct prediction over all, it gives the accuracy score of the model within each target class. For a two-class problem, the first row of the comfusion matrix will contain the values of the *true negatives* and *false negatives*, and the second row will show the values of *false positives* and *true positives*.

Finally, we use the area ratio in the *cumulative gains chart* as a way of comparing the performance of our models. Such a chart has three curves which respectively represent the performance of a perfect classifier, a random classifier (baseline) and our model, with the horizontal axis representing the percentage of total data and the vertical axis representing the cumulative percentage of target data. The area ratio is defined

as

$$\text{area ratio} = \frac{A_{model} - A_{baseline}}{A_{perfect} - A_{baseline}}, \tag{26}$$

where a high area ratio would represent a good model.

2.5. **Description of the data.** For the classification case, we use a data set of credit card holders from a bank in Taiwan in 2005, which is available on the UCI website[1]. The original data set contains 30.000 observations of whether the credit card holders default on their payment or not, and uses 23 explanatory variables as the predictors for this binary outcome, where in total 22% of the target variables are observations of defaults.

We have chosen a couple of different avenues when it comes to preprocessing the data set, and in each case trained our models on it. The categorical features in the data set are those which specify gender, education and marital status, as well as payment status of previous months, with the remaining features being continuous. As explained on the UCI website, the history of previous payments are categorized from -1 to 9, with -1 representing that the credit card holder has payed duly, and $1-9$ representing the number of months delay of payment. Because this feature has been categorized in this way, we have chosen to treat it as a categorical feature in one case, and as a continuous feature in another, to see which way of treating it gives the most accurate model. Categorical features have in both cases been one hot encoded, and continuous features have been min-max scaled to the range $[-1, 1]$. In our preprocessing of the data we have found some outliers in the explanatory variables which specify gender, education and marital status, and have opted to simply delete these entries. Since the data set we are operating on contains many more observations of non-defaults than defaults, we have also tried training our models on data in which we have randomly removed non-default entries from the training data, such that the default/non-default ratio is 1:1 on the training data. In the case where the data was balanced, we have (prior to balancing) split it randomly into training data and test data, with 10% being test data, and in the case of not balancing, 20% was used for testing.

For the regression case we used the Franke function[7] to generate a data set. We used a grid size of 20×20 and a random generated noise with standard deviation 0.2, and compared this to the results from our previous project[2]. Because of an unnoticed bug, the noise in the Franke function is only generated randomly along one axis, which means that the meshgrid gets the same noise value along one of the axes. This results in a surface with smooth ridges, as we can see in figure 3.7. The original plan was to have random noise along both axes in the meshgrid, but the bug was noticed too late in the analysis. This should however have little effect on the analysis of the machine learning methods, since the data was generated in the same way in our previous project, which is what we compare our results with.

2.6. **Benchmarks.** For logistic regression, we have constructed a test case where we compare the performance of our own implementation with Scikit Learn's *LogisticRegressionCV* function on Scikit's breast cancer data set, by calculating the accuracy in both cases. The average difference in accuracy is in the order of $\epsilon = 10^{-4}$. In the case of classification with neural networks, we tested our code against Scikit Learn's *MLPClassifier*, using the same breast cancer data set. Our code gave on average a better classification accuracy and a better area ratio compared to the results from Scikit Learn, and the difference was in the order of 10^{-1}.

We also constructed a test case for regression, using the Franke data set with a 100×100 grid and a noise of 0.1, where we benchmarked our neural network code against Scikit Learn's *MLPRegressor*.

[1]UCI website: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

With identical parameters passed into both our own code and the Scikit Learn's method, we obtained a difference in MSE and R^2 score in the order of 10^{-2}.

2.7. **Source code.** The source code of this project is written in Python, and can be found in the GitHub repository at `https://github.com/ejhusom/FYS-STK4155/tree/master/project2/`. The repository contains our source code in the folder `src`, which consists of the following files:

- `breastcancer.py`: Preprocessing of the breast cancer data
- `creditcard.py`: Preprocessing of the credit card data.
- `franke.py`: Preprocessing of the Franke data.
- `logistic_classification.py`: Functions for analyzing the logisitic classification.
- `main.py`: Main entry to running the analysis.
- `nn_classification.py`: Functions for analyzing the neural network when applied to classification problems.
- `nn_regression.py`: Functions for analyzing the neural network when applied to regression problems.
- `pylearn`: Module containing our code for various machine learning methods, including logistic regression and neural networks.

3. Results

3.1. **Classification.** For logistic regression, we see from table 3.1 that the way of preprocessing the data which gave the best results was to one hot encode all categorical features and balance the training data such that it contained a 1:1 ratio of the number of defaults and non-defaults, where we have deemed the number of true positive and negative classifications as the most important error metric. The results that follow for logistic regression are based on this preprocessing format of the data. We have done a grid search of the optimal values of the learning rate and number of epochs in the SGD, shown in figure 3.1. The parameters have been evaluated based on the over all accuracy score produced by the model with the respective configuration, which we found to be 0.82 with a learning rate $\eta = 10^{-3}$ and 500 SGD epochs. Table 3.2 shows the confusion matrix of the models predictions on the test data with this configuration, where it predicted true positives with an accuracy of 0.35 and true negatives with 0.96 accuracy.

Table 3.1. Results from classification of the credit card data with logistic regression, in terms of accuracy, area ratio and true positive/negative classifications. Description of the preprocessing methods:
1) One hot encoding of gender, education and marital status.
2) One hot encoding of gender, education, marital status and payment history.
3) Balanced training set with 1:1 ratio of defaults and non-defaults, and one hot encoding of gender, education and marital status.
4) Balanced training set and one hot encoding of gender, education, marital status and payment history.

Preprocessing method	Accuracy		Area ratio		True positive	True negative
	Train	Test	Train	Test	Test	
1	0.80	0.81	0.45	0.42	0.24	0.97
2	0.82	0.82	0.54	0.51	0.34	0.96
3	0.81	0.80	0.46	0.44	0.26	0.97
4	0.82	0.82	0.53	0.51	0.35	0.96

TABLE 3.2. Confusion matrix for the predictions of our optimal classifiers on the credit card test data. Label 1 represents default payment, and 0 non-default.

Logistic regression

		Prediction	
		0	1
Actual	0	0.96	0.04
	1	0.65	0.35

Neural network

		Prediction	
		0	1
Actual	0	0.92	0.08
	1	0.36	0.64

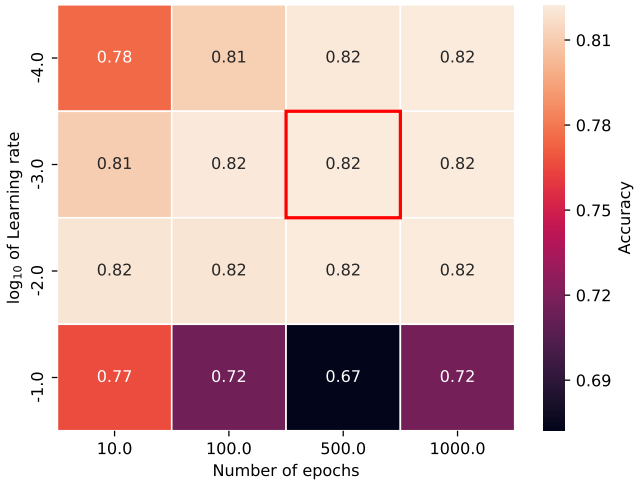

FIGURE 3.1. Heat map of the grid search for best learning rate and no. of epochs with logistic regression on the credit card data. Each of the data points have been found using k-fold cross validation with $k = 5$ for a better estimation of the accuracy.

For our neural network classifier, we found an optimal learning rate for the model to be $\eta = 10^{-1}$, shown in figure 3.3. The figure also shows that the best way of data preprocessing is to one hot encode all categorical features and balance the training data in terms of number of defaults/non-defaults. Based on these parameters, we have done a grid seach of the optimal number of hidden layers and nodes per layer in the neural network, shown in figure 3.4, where we see that the optimal configuration is three hidden layers with 80 nodes each. Table 3.2 shows the confusion matrix of our final model, trained with the parameters found above and 10.000 SGD epochs, giving a true positive rate of 0.64, true negative rate of 0.92 and an accuracy score of 0.86 on the test data.

In table 3.1 we have listed the results from our best classifiers and for comparison those of I-C Yeh, C-h Lien[1][p.5]. In the case of logistic regression, we see that out results are very similar in terms of accuracy (0.81 vs 0.82 on the test data), though we achieve a higher area ratio with our model (0.51 vs 0.44 on the test data). For neural networks, our optimal model has an accuracy score of 0.86 on the test data, while Yeh and Lien's is 0.83. We also achieve a higher area ratio with our model (0.65 vs 0.54 on the test data). The gain charts of our classifiers are shown in figures 3.2.

3.2. **Regression.** For our regression case, where we used a data set generated by the Franke function, the initial analysis of the learning parameter is shown in the left plot in figure 3.5, which were found

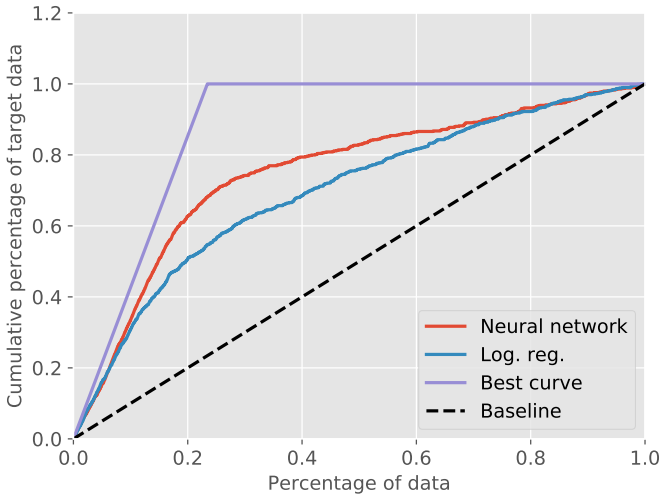

FIGURE 3.2. Gain curves of our classifiers on the credit card test data. The blue curve shows the results for logistic regression (area ratio 0.51), the red curve shows the results for neural network (area ratio 0.65).

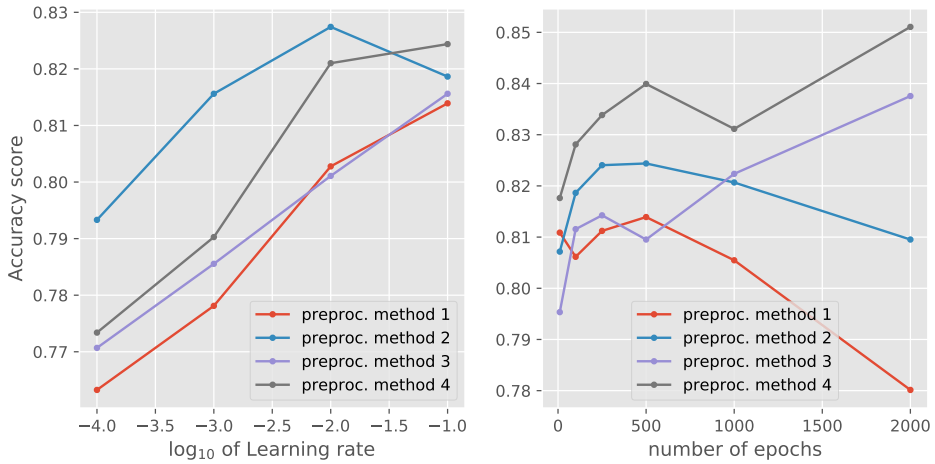

FIGURE 3.3. Accuracy score when testing a neural network with varying learning parameter and number of epochs, using 3 layers with 100 nodes in each. The best performing learning rate (in terms of accuracy score) were selected from the left plot, which were $\eta = 10^{-1}$, and this value was used when producing the plot to the right. In the initial analysis in the left plot we used 100 epochs when training the network. The preprocessing methods are as defined in the caption of table **??**.

to be 10^{-2} when using the ReLU as activation function. The sigmoid function was also tested for the same values of the learning rate, excluding 10^{-1} because the gradient diverged, and we were not able to

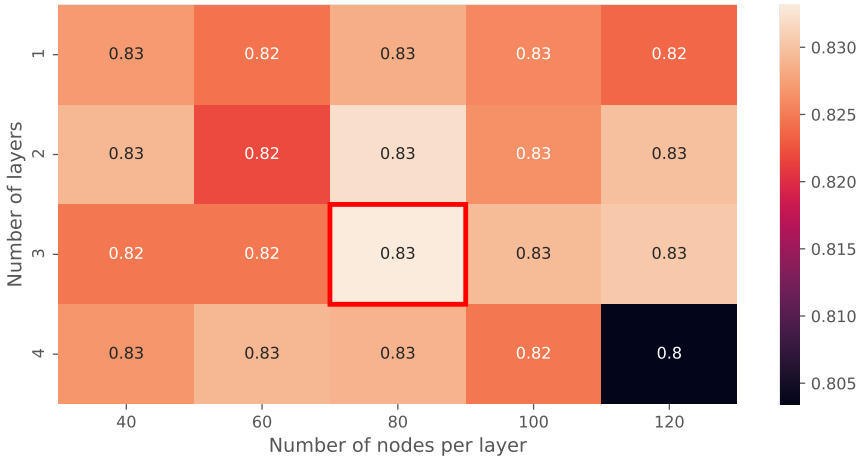

FIGURE 3.4. Heat map of grid search for configuration of number of layers and nodes per layer when using a neural network. The highest accuracy score obtained is marked with red rectangle. In this grid search we have used 100 epochs in the SDG and a learning rate $\eta = 10^{-1}$.

TABLE 3.3. Classification results from logistic regression and classification.

	Our models				I-C Yeh, C-h Lien			
Method	Accuracy		Area ratio		Accuracy		Area ratio	
	Training	Test	Training	Test	Training	Test	Training	Test
Logistic regression	0.82	0.81	0.53	0.51	0.80	0.82	0.41	0.44
Neural Network	0.98	0.86	0.99	0.65	0.81	0.83	0.55	0.54

get a result. We then performed an analysis of how the mean squared error behaved as a function of number of epochs, as shown in the right plot in figure 3.5. Figure 3.6 shows a grid search of the optimal configuration of number of layers and number of nodes in each layer. The best performing configuration from this analysis is then used in figure 3.7, where we see a plot of the generated data as a 3D surface, with the neural network prediction as a wiregrid on top. In this case we used 2000 epochs to produce the results, because we observed slightly better results when increasing the number of epochs. Table 3.4 shows the MSE and R^2 score for our neural network, compared with the regression methods which we used in our previous project[2]. In all of the figures and tables mentioned in this section, the MSE and R^2 score is measured on the test set of the data, which consists of 20% of the full data set.

In figure 3.8 we see a plot of how the performance metrics MSE and R^2 behaves as a function of the grid size of our Franke data set which is sent through our neural network. The figure caption contains the parameters used when creating the models.

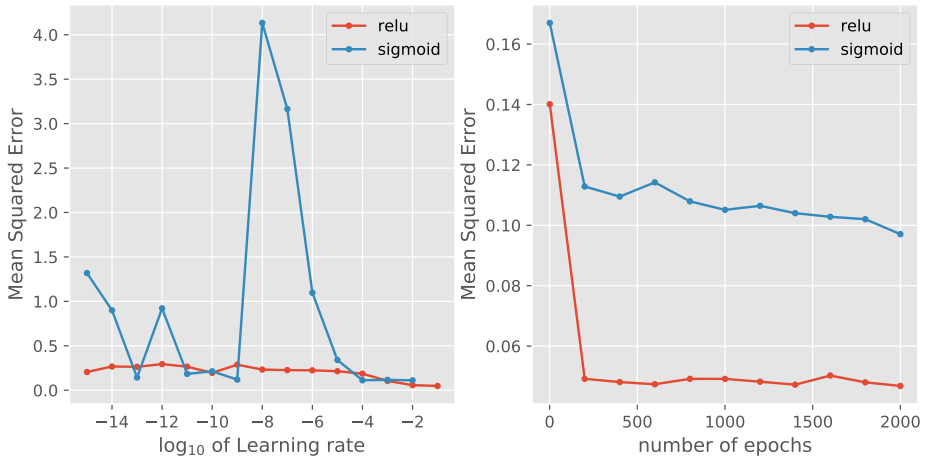

FIGURE 3.5. MSE of the test set when testing a neural network with varying learning parameter and number of epochs on the Franke data set, using both ReLU and sigmoid as activation function. The best performing learning rate was found to be 10^{-1} in the left plot, and this learning rate was used when analyzing number of epochs in the right plot. In all of these cases we used two layers with 100 nodes in each.

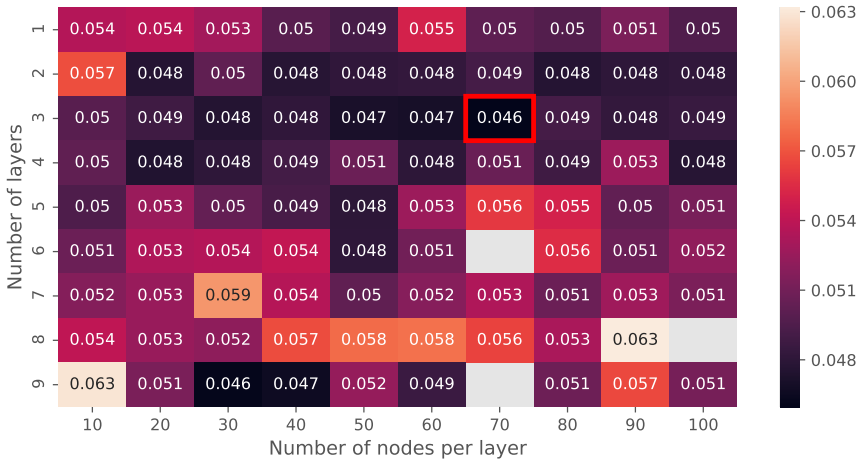

FIGURE 3.6. Heat map of grid search for configuration of number of layers and nodes per layer when using a neural network with learning rate of 10^{-1} and 500 SGD epochs. The lowest MSE obtained on the test set is marked with red rectangle. The blank grey boxes indicate cases where the gradient exploded during the training of the network, and no result were obtained.

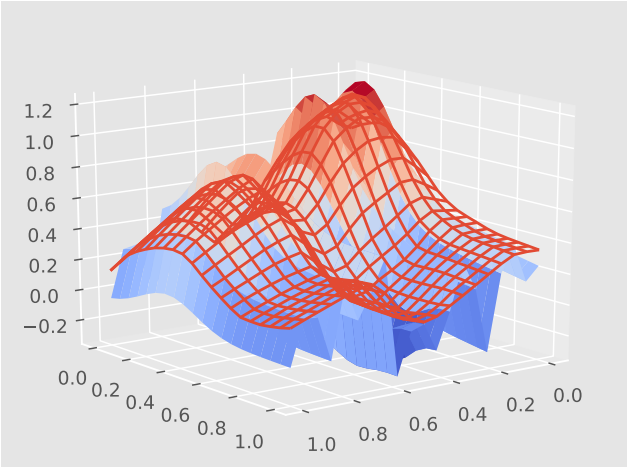

FIGURE 3.7. The Franke data (solid surface), with gridsize 20×20 and a noise of $\epsilon = 0.2$, plotted together with the prediction from the neural network (wiregrid), using 3 hidden layers with 70 nodes each on a run of 10000 epochs.

TABLE 3.4. Comparison of MSE and R^2 score (measured on the test set) between the three different regression methods used in our previous project[2] and the neural network, when used to model data generated from the Franke function.

Method	MSE	R^2
OLS	0.0136	0.819
Ridge	0.0139	0.877
Lasso	0.0183	0.728
Neural network	0.0415	0.631

4. Discussion

4.1. **Classification.**

4.1.1. *Logistic Regression.* From our results in the case of logistic regression, we see from table 3.1 that our models performed equally well in terms of accuracy irrespective of the different preprocessing cases, with accuracy scores on the test data in the range $0.80 - 0.82$ in all cases. However, accuracy as a performance metric in a binary classification problem is obviously not a good way to evaluate how well a certain classifier performs, especially when 78% of the outcomes are in the same category. The importance of preprocessing the data correctly becomes more apparent when looking at the rate of true positive and true negative classifications in the same table (3.1). Here we see that, for instance method 1) and 4), where one data set had been balanced in terms of equal outcomes in the training set and the other had not, have nearly identical accuracy scores, but the difference in the models' ability to predict true positives differ by over 10%. With only 22% of the observations in the data belonging to the under

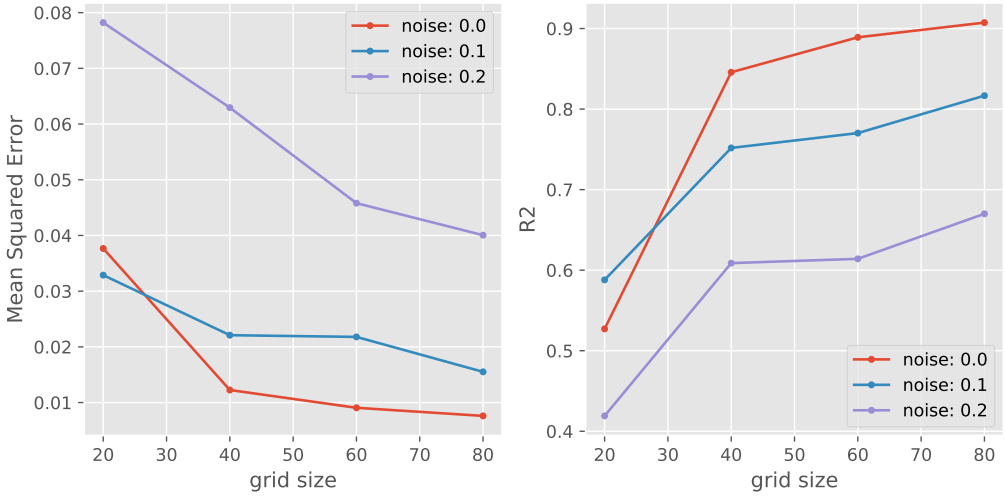

FIGURE 3.8. MSE and R^2 score (measured on the test set) as a function of the grid size of the Franke function, for three different values of noise introduced to the data set, when a neural network with 3 hidden layers of 70 nodes each, and a learning parameter of 10^{-1}, is applied to the data set, over 200 epochs.

represented class, a model which trained on this data is easily over fitted to represent the numerous class of outcomes, and our results in these cases are a classical example of this. In our balancing of the data, we have opted to randomly delete cases of the numerous class, however, in a scenario with a smaller data set, it may instead be better to duplicate observations belonging to the under represented class.

4.1.2. *Neural networks.* When using neural networks we started the analysis with tuning the learning parameter and the number of epochs, as we did with logistic regression. The training of the network was very computationally expensive, which prevented us from performing both a cross-validation and a grid search for the optimal parameters. The left plot in figure 3.3, indicates that a learning rate of 10^{-1} is the most stable choice, even though a higher accuracy score was obtained with preprocessing method 2 and a learning rate of 10^{-2}. As stated above, the accuracy score is not a very good performance metric, which somewhat undermines the result of these results, but it is nevertheless an indicator of what might work best. In the right plot of figure 3.3, we see a general trend of higher accuracy score when increasing the number of epochs for the two data sets that are balanced, while the unbalanced data sets has a decreasing accuracy score. Based on these results, we chose to use the data set with preprocessing method 4 in the next step of our analysis, because it generally had obtained the highest accuracy score. To find the best configuration of number of hidden layers and the amount of nodes in each layer, we performed a grid search, as shown in figure 3.4. A problem with this grid search is that we do not necessarily need to have the same amount of nodes in every layer, and we might miss layer configurations which are superior to the ones we tested. Doing a grid search of layer configurations with varying amount of nodes is too costly in terms of computational resources, but we might have obtained better results by doing a random search for more complex layer configurations. Again we face the problem of accuracy score not being the best performance metric, and we can see that the score has only slight variations depending on what configuration of hidden layers that we choose. Even so, the configuration with 3 layers with 80 nodes,

which gave the highest test accuracy score, falls into the range of recommended values the acknowledged machine learning textbook by Géron[6][pp. 290-291], and we chose to use that for our final model.

When comparing our results with those of Yeh & Lien in table 3.1, we found that our results for logistic regression in terms of accuracy were very similar, however our classifier achieved a higher area ratio (0.51 vs 0.41). Also with our neural network classifier, out model achieves a higher area ratio than Yeh & Lien (0.65 vs 0.54). From the same table we can also see that the area ratio on the training data is 0.99, which may suggest that our neural network is slightly over fitted on the training data. As to why our classifiers perform better in both cases it not easy to say, as Yeh & Lien do not specify the parameters of their models, nor do they give a description of how they have treated the data prior to training.

4.2. **Regression.**

4.2.1. *Learning rate and number of epochs.* The analysis of the performance of neural networks applied to a regression problem started with a search of the optimal learning rate, as shown in the left plot of figure 3.5. We tested both ReLU and sigmoid as activation functions, but as the plot shows, the ReLU was the most stable of the two, and is also the one that is commonly used for regression problems. The results also shows that the lowest MSE were obtained by using a learning rate of either 10^{-1} or 10^{-2}, the former giving slightly better performance. The plot to the right in the same figure shows how much the MSE decreases when we increase the number of epochs, and it indicates that we do not gain much, in terms of MSE, by using more than 500 epochs. When doing this initial analysis, we used two layers with 100, which were chosen on the basis of heuristics in the field of machine learning[6].

4.2.2. *Hidden layers configuration.* In order to find the optimal number of layers and number of nodes in each layer, we used a grid search, as shown in figure 3.6. In this analysis we used ReLU as activation function with a learning rate of 10^{-1}, as we previously had found the give the best results, and the heatmap indicates that using 3 layers with 70 nodes in each layer gives the lowest MSE, among the options that we tested. We face the same challenge here as we did when using the neural network for classification, that the optimal configuration might be to have a varying number of nodes in the different layers, but again, this would be a too complex analysis to perform using a grid search. Also, we could try to improve our model by using modified versions of the ReLU function, for example the Leaky ReLU, or even to have different activation functions in different layers. We refrained from exploring these possibilities in order to keep the analysis manageable, but it might have improved the performance of our model.

4.2.3. *The tuned model.* When using the parameters that were found to give the best results, we obtained the model shown in figure 3.7, where the wiregrid is our neural network prediction, and the coloured surface is the original dataset. The resulting MSE and R^2 score is shown in table 3.4, and we can see that the neural network performs significantly worse than the regression methods OLS, Ridge regression and Lasso regression when applied to the same data set. one of the reasons for this might be sub-optimal tuning of the hyperparameters, specifically the learning rate, the number of epochs and the configuration of hidden layers. In our analysis we split the data set into a training set and a set test with a 80%/20% split, but did not utilize any resampling techniques to optimize the tuning, as we did with logistic regression. As seen in figure 3.5, there was little difference between choosing a learning rate of 10^{-1} or 10^{-2}, and the MSE values were obtained on a single training run with a certain random seed. If we had chosen to use the latter learning rate instead of the former, we might have ended up with another configuration of the hidden layers, and possibly better performance metrics. Different seeds also affect the outcome, so by using for example cross-validation we could have tuned our parameters with more certainty.

Another reason for the relatively poor results of our neural network seems to be the small data set size. As we can see in figure 3.8, where the grid size of the Franke function is plotted against the obtained MSE and R^2 score of the test set. We can clearly see that we obtain a better performance of our model when we increase the amount of data that is fed to the neural network. We also observe that the test MSE and R^2 is also greatly affected by the noise in the data set, and might indicate that the high test MSE of the neural network is a result of overfitting.

The size of the minibatches is another parameter of the model, which we have chosen to have constant through all our analysis. We might have seen improvement by increasing or decreasing the batch size, but because the number of parameters to tune when using a neural network is significant, we chose to leave the batch size at 100, in order to reduce the complexity of the analysis. We could also have introduced so-called regularization[3][p. 398] to our neural network for both regression and classification, but initial testing of this method did not improve our results, so we chose to not include that in our analysis. Furthermore, it is common for stochastic gradient descent implementations to include an adaptive learning rate[2], which we also tried in the early stages of our code development, but because our implementation of it seemed to give no significant improvement, which might be caused by a bug, we chose to stick with a constant learning rate for the sake of simplicity.

5. Conclusion

In this project we study the performance of two classification methods, logistic regression and artificial neural networks, and compare their predictive performances. The prediction accuracy of the two methods have been found to be similar, but when using confusion matrices and area ratio of the cumulative gain curve as accuracy metrics, we find that neural networks generally perform better than logistic regression. We also study ways of preprocessing the data, where we find that balancing the data prior to training can give a much better result. On this balanced data, we find the best parameters for the neural network to be a learning rate of $\eta = 10^{-1}$, three hidden layers with 80 nodes each and 10.000 SGD epochs. This classifier has an accuracy score of 0.86 and an area ratio of 0.65 (table 3.1), and 0.64 of the classifications were true positives and 0.92 true negatives (table 3.2).

We also study the performance of neural networks when applied to a regression problem, and compare this with the performance of common regression methods. We find that the neural network performs best when using the ReLU function as an activation function, and a learning rate of $\eta = 10^{-1}$. The lowest MSE of 0.0415 is obtained when using three hidden layers with 70 nodes each, while the Ordinary Least Squares method give an MSE of 0.0136 when applied to the same data set (table 3.4). Our results also indicate that the neural network performs better for larger data sets.

[2]Scikit Learn's documentation on stochastic gradient descent: https://scikit-learn.org/stable/modules/sgd.html

References

[1] I-C Yeh and C h Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications.*, 2009.
[2] A. Sexton and E. Husom. Regression analysis and resampling methods. 2019.
[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2009.
[4] A. C. Faul. *A Concise Introduction to Numerical Analysis*. CRC Press, 2016.
[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks.
[6] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.
[7] R. Franke. *A Critical Comparison of Some Methods for Interpolation of Scattered Data*. Naval Postgraduate School, California, 1979.